Estratégia para Proteção de Arquivos Compactados Corrompidos por Surtos de Erros

Zaqueu Cabral Pereira, Richard Demo Souza e Marcelo Eduardo Pellenz

***Resumo*— Este trabalho apresenta uma estratégia de proteção desigual de erros para minimizar o impacto dos surtos de erros no armazenamento e na transmissão de arquivos compactados. A estratégia é feita para o algoritmo de compressão de dicionário adaptativo LZSS. Através de simulações computacionais foram investigadas quais classes de informação do LZSS são mais sensíveis aos surtos de erros. Neste trabalho propomos uma estratégia de proteção desigual que limita a propagação de erros no processo de descompressão. A estratégia de proteção proposta foi validada através de resultados numéricos, usando arquivos de texto padrões da base Calgary Corpus.**

***Palavras-Chave*— Proteção desigual de erros, compressão de dados, LZSS, codificação de canal.**

***Abstract*— This work shows an unequal error protection strategy for minimising the impact of error bursts in the storage or transmission of compressed data. The strategy is designed for the LZSS algorithm, which is a dynamic dictionary compression method. Through extensive computer simulations we investigate which information classes within LZSS are more sensitive to burst errors. Based on these results we propose an unequal error protection strategy that greatly reduces the error propagation during the decompression process. Numerical results using text files from the Calgary Corpus common database are presented.**

***Keywords*— Unequal error protection, data compression, LZSS, channel coding.**

I. INTRODUÇÃO

Para um melhor aproveitamento, dispositivos de armazenamento e sistemas de comunicações requerem a utilização de técnicas eficientes de compressão de dados. No caso de informação textual [1], banco de dados [2] e alguns tipos de imagens [3], [4] , os algoritmos de compressão sem perdas do tipo Lempel-Ziv são os mais utilizados [5]–[9]. Contudo, dados compactados são muito sensíveis a erros, um único erro no arquivo compactado pode propagar para diversos erros no arquivo descompactado.

Sistemas de armazenamento de dados com grande capacidade e acesso rápido, como dispositivos semicondutores, magnéticos e óticos, são suscetíveis a erros no processo de leitura e gravação, onde surtos de erros podem ocorrer devido a regiões defeituosas [10]–[12]. Surtos de erros também podem ocorrer devido a um desvanecimento profundo em canais de comunicação sem fio [13], [14]. Desse modo, nota-se que a falta de resistência a erros dos arquivos compactados requer a aplicação de técnicas de correção de erros [15], [16] para o armazenamento ou transmissão confiável dos dados compactados. Essa proteção reduzirá o efeito da propagação de erros durante o processo de reconstrução. Porém, o objetivo da compressão de dados, que é a redução do tamanho dos arquivos, pode ser comprometido. Como a utilização de esquemas de proteção uniforme (EEP) pode prejudicar o efeito da compressão, é de suma importância o desenvolvimento de métodos que apliquem proteções mais eficazes aos dados compactados.

Nas técnicas desenvolvidas por Lempel e Ziv [5], [7], o dado compactado é separado em diferentes classes, algumas dessas com uma importância maior que outras no processo de descompressão. Tais classes necessitam de uma proteção mais rigorosa contra erros. Este é um caso típico onde esquemas de proteção desigual de erros (UEP) podem ser usados com sucesso [17]–[19]. Importantes esquemas UEP para dados compactados usando o algoritmo LZSS foram propostos em [20], [21].

O algoritmo LZSS [6] é uma variação do LZ77 [5], com um melhor desempenho. É um algoritmo muito utilizado para compressão de textos, pois tem uma grande eficiência para esse tipo de informação [1], [6]. Por esta razão adotamos o algoritmo LZSS para realizar o estudo da estratégia de UEP. O LZSS tem aplicações práticas nos softwares de compactação PKZip, Zip, LHarc, ARJ, gzip e no padrão PNG.

Neste trabalho investigamos o impacto de surtos de erros em arquivos de texto compactados usando o algoritmo LZSS. Baseados nesses resultados é proposta uma nova estrutura para os arquivos compactados que visa uma proteção maior nas classes mais sensíveis, com uma redundância particular para cada classe. O esquema proposto é implementado usando códigos Reed-Solomon, que são altamente eficientes na correção de surtos de erros [15], [16].

Resultados de simulações mostrando a eficiência do esquema proposto, sua comparação com proteção uniforme (EEP) e com outros esquemas UEP também são apresentados. As simulações apresentadas nesse trabalho usam os arquivos da base de dados Calgary Corpus [22], que é utilizada como padrão para testes de algoritmos de compressão.

Um esquema de proteção desigual com códigos Reed-Solomon para dados compactados com LZSS foi implementado em [20]. Esse esquema difere-se do método introduzido neste artigo no sentido que não garante proteção total contra propagação de erros, pois aplica proteção em apenas uma das classes. Para validação são realizados testes apenas para um arquivo utilizando uma métrica símbolo a símbolo, que não é apropriada. Neste trabalho, são realizados testes com diversos arquivos para validação do esquema, utilizando como métrica a distância de Levenshtein [27], [28], a qual é mais indicada

Marcelo Eduardo Pellenz, PPGIA, PUCPR, Curitiba, Paraná, Brasil, E-mail: marcelo@ppgia.pucpr.br.

Richard Demo Souza e Zaqueu Cabral Pereira, CPGEI, UTFPR, Curitiba, Paraná, Brasil, E-mails: richard,zaqueu@cpgei.cefetpr.br

para arquivos de texto.

Outro esquema de UEP para LZSS foi proposto em [21]. Tal esquema é baseado em um tipo de código de repetição das classes mais sensíveis. Os códigos Reed-Solomon utilizados neste trabalho, mesmo sendo mais exigentes computacionalmente, são mais eficazes para correção de surtos de erros com uma inserção de redundância adicional muito menor que os códigos de repetição utilizados em [21].

O esquema proposto neste trabalho pode ser aplicado em casos onde a entrega de textos compactados ou arquivos de dados com poucos erros seja possível. Nesses casos alguns erros locais poderiam ser aceitáveis, enquanto erros globais (como os causados devido à propagação no processo de descompressão) seriam inaceitáveis. Exemplos de tais casos são a compressão de grandes volumes de dados [23] e a transmissão de texto em tempo-real em canais sem fio [24], [25].

O restante deste artigo está organizado como segue. Na Seção II apresentamos rapidamente o algoritmo de compressão de dados baseado em dicionário adaptativo, LZSS. Na Seção III investigamos, através de simulações computacionais, o impacto do surto de erros na descompressão de arquivos compactados com LZSS. Uma estratégia UEP desenvolvida para o algoritmo LZSS é introduzida na Seção IV. Simulações computacionais mostrando o desempenho do esquema UEP proposto são apresentadas na Seção V. Por fim, na Seção VI fazemos alguns comentários finais.

II. Algoritmo LZSS

O algoritmo LZSS [6] é um tipo de método de dicionário adaptativo variante do LZ77 [5]. No método LZSS, o dicionário é simplesmente uma parte da seqüência previamente codificada. O codificador examina a seqüência de entrada através de uma janela deslizante como visto na Figura 1. A janela consiste em duas partes, um *buffer* de busca, ou dicionário, que contém uma parte da seqüência recentemente codificada, e um *look-ahead buffer* que contém a próxima parte da seqüência a ser codificada. Na Figura 1, o *buffer* de busca contém oito símbolos, enquanto o *look-ahead buffer* contém sete símbolos.

Fig. 1. Exemplo de uma janela deslizante do codificador LZSS.

A janela deslizante utilizada no LZSS é basicamente a mesma do LZ77. Porém, para evitar o problema do LZ77 de ineficiência na codificação, faz-se uma mudança muito importante no algoritmo. A codificação do LZ77 gera na saída uma tripla (o, m, c). Nesta, (o) é chamado *offset* e é a distância do ponteiro até o *look-ahead buffer*. O *match* (m) é o número de símbolos consecutivos no dicionário que são idênticos aos símbolos consecutivos no *look-ahead buffer*, iniciando com o primeiro símbolo referenciado no ponteiro, ou seja, o *match* especifica o comprimento da seqüência. O LZ77 ainda usa um terceiro elemento que é o primeiro símbolo do *look-ahead buffer* subseqüente à frase: (c). Mesmo quando o algoritmo não encontra uma seqüência de símbolos repetidos no dicionário, ela envia a tripla com os elementos *offset* e *match* zerados, e um caractere sem codificação, tornando o LZ77 ineficiente nesta situação.

O LZSS também utiliza os elementos *offset*, *match* e caractere. Porém, quando o algoritmo não encontra seqüências de símbolos repetidos no dicionário, não é mais gerada a tripla na saída com os dois primeiros elementos zerados. O LZSS usa um bit simples como prefixo, chamado de *flag* (f), para definir que tipo de palavra-código será gerada. O *flag* assume o valor 0 se não existe nenhuma seqüência de símbolos repetidos no dicionário, ou seja, gera uma saída (f, c) para um caractere não codificado. Quando existe uma seqüência a codificar, o *flag* assume o valor 1, que vem acompanhado do *offset* e do *match* (f, o, m). A Equação 1 define a composição da palavra-código C_i, que é a i-ésima palavra-código gerada pelo codificador, no método LZSS:

$$C_i = \begin{cases} (f_i, o_i, m_i) & \text{para } f_i = 1 \\ (f_i, c_i) & \text{para } f_i = 0 \end{cases} . \qquad (1)$$

O arquivo compactado utilizando o algoritmo LZSS pode ser armazenado ou transmitido de acordo com a estrutura vista na Figura 2, onde f denota os *flags*, c denota os bits de caractere, m corresponde aos bits de *match*, e o aos *offsets*.

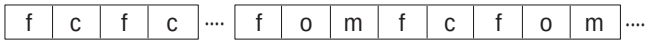

Fig. 2. Estrutura dos dados compactados pelo algoritmo LZSS.

III. Impacto dos Surtos de Erros no Processo de Descompressão

Durante a descompressão do LZSS, se o *flag* indicar um caractere não-codificado, o algoritmo simplesmente copia o próprio caractere na seqüência decodificada. Se o *flag* indicar uma palavra-código, o algoritmo copia o símbolo indicado pelo ponteiro (*offset*) e todos os símbolos que seguem referenciados pelo comprimento do *match*. Desse modo, se erros são introduzidos durante o armazenamento ou transmissão do arquivo compactado, seus efeitos na descompressão podem ser muito grandes, dependendo do tipo de informação que será afetada. Por exemplo, se os bits de *match* são afetados, o tamanho do arquivo reconstruído será alterado em relação ao original. Porém, se os bits de caractere são afetados, ocorrem uma propagação de erros quase desprezível.

Para avaliar o impacto de um surto de erros ao longo do arquivo compactado com o algoritmo LZSS, foram feitos testes com arquivos de texto de uma base de dados comum, o Calgary Corpus [22]. Arquivos da mesma base de dados foram utilizados em [18]–[21] para analisar esquemas de proteção desigual de erros para variantes Lempel-Ziv. Alguns desses métodos serão tratados nas próximas seções. A Tabela I mostra os parâmetros resultantes da compressão do arquivo 'paper4.txt', o qual tem um tamanho original de 13286 bytes.

O algoritmo de compressão LZSS foi implementado da maneira original [6], com os parâmetros seguintes: 8 bits para um caractere não-codificado, 1 bit para os *flags*, 16 bits para os *offsets* e 4 bits para os símbolos de *match*.

TABELA I

PARÂMETROS DO ARQUIVO COMPACTADO A PARTIR DO ARQUIVO 'PAPER4.TXT'.

Parâmetros	
Tamanho do arquivo (bytes)	7164
Tamanho do arquivo (bits)	57312
Taxa de Compressão	46,1%
Número de bits de *flag*	3796
Número de bits de caractere	14912
Número de bits de *offset*	30880
Número de bits de *match*	7724

Para demonstrar o impacto de um surto de erros no processo de descompressão, foi utilizado um surto de erros de 48 bits, em diferentes posições do arquivo compactado. O surto de erros é aplicado para intervalos consecutivos de 48 bits, semelhante a regiões deficientes em um dispositivo de armazenamento de dados [11], [15], [16], [26], ou a um canal sem fio com grande desvanecimento [13], [14]. Na Figura 3 é mostrada a taxa de erro de símbolo (SER) no arquivo reconstruído como uma função da posição do surto de erros. A SER é definida como a distância de Levenshtein pelo número de símbolos na mensagem transmitida. A distância de Levenshtein é o número mínimo de inserções, eliminações e substituições de símbolos requerido para transformar a mensagem decodificada na mensagem original transmitida [27]. De acordo com [28], a distância de Levenshtein é uma medida mais apropriada que uma simples comparação símbolo a símbolo, a qual não detectaria o efeito de possíveis inserções e eliminações.

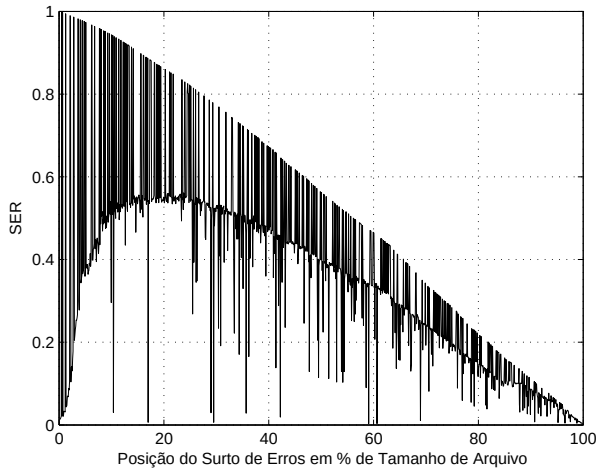

Fig. 3. Impacto de um surto de erros de 48 bits no processo de descompressão, em termos da SER como função da posição do surto de erros para o arquivo 'paper4.txt'.

Na Figura 3 pode-se ver que a propagação de erros é mais grave quando o surto de erros está localizado no início do arquivo compactado. O valor médio da SER na Figura 3 é de $36,82\%$.

Uma forma simples para combater a propagação dos erros é aplicar um código corretor em todo arquivo compactado que garanta a proteção para um surto de erros com comprimento l. Porém, dessa maneira dois objetivos serão contrapostos no processo. A codificação de fonte ou compressão de dados remove a redundância; enquanto a codificação de canal introduz redundância adicional. Com essa estrutura, a redundância aplicada ao arquivo compactado torna inviável esse método, pois a taxa de compressão seria muito prejudicada.

Considere uma nova estrutura que permita uma análise melhor de cada uma das classes dos dados compactados com LZSS. Nessa nova estrutura os bits são organizados em quatro classes diferentes, como mostra a Figura 4, onde **F**, **C**, **M** e **O** correspondem aos bits de *flag*, de caractere, de *match* e aos de *offset*, respectivamente.

Fig. 4. Estrutura em classes para o arquivo compactado com LZSS.

Após simulações com os dados compactados e organizados na estrutura da Figura 4, conclui-se que essa estrutura é consideravelmente menos sensível a surtos de erros que a estrutura original do LZSS, mostrada na Figura 2. Na Figura 5, tem-se a taxa de erro de símbolo (SER) no arquivo descompactado em função da posição do surto de erros, considerando a nova estrutura. A partir dessa figura, pode-se notar que a propagação de erros é muito maior quando os surtos são aplicados nas posições que têm uma grande quantidade de *flags* e *matches* (primeiro e segundo picos, respectivamente). Considerando essa nova estrutura para o arquivo compactado, a SER média depois da descompressão é de $10,68\%$. Um decréscimo considerável quando comparado com o caso da Figura 3, a qual usa a estrutura regular do arquivo compactado com LZSS como visto na Figura 2, e onde a SER média é de é $36,82\%$.

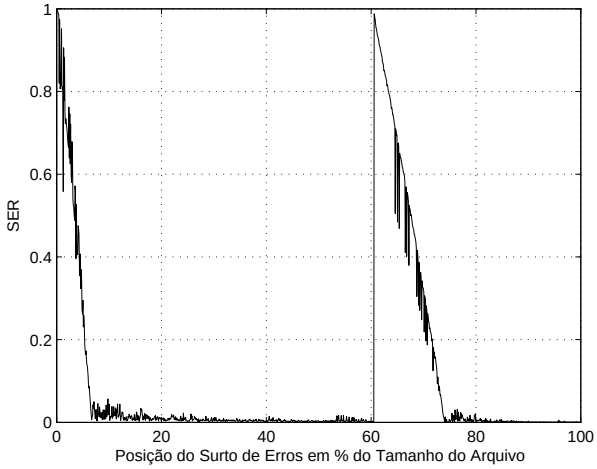

Fig. 5. Impacto de um surto de erros de 48 bits no processo de descompressão, em termos da SER e como uma função da posição do surto de erros para o arquivo 'paper4.txt', considerando a nova estrutura do arquivo compactado.

Utilizando a estrutura proposta na Figura 4, pode-se analisar o efeito de um surto de erros para cada classe separadamente. A Figura 6 mostra o efeito do surto de erros no processo de descompressão, quando o surto de erros de 48 bits é aplicado em cada uma das quatro classes. Analisando a figura, onde as curvas dos caracteres e dos *offsets* são quase coincidentes, pode-se ver que as classes mais sensíveis são os *flags* e os *matches*. Se um bit de *flag* é corrompido, então um caractere não-codificado é substituído por uma palavra-código, ou vice-versa. Como resultado, os *offsets* apontarão para uma posição inicial errada, e os *matches* indicarão um comprimento da seqüência errado para ser usado no dicionário. Assim, erros nos *flags* irão corromper o tamanho do arquivo, gerando um arquivo descompactado com um tamanho diferente do arquivo original. Um erro nos *matches* também gera um comprimento da seqüência errado no dicionário e causa erros graves no arquivo descompactado. Em contra partida, os erros nos *offsets* e nos caracteres causam um impacto muito pequeno na descompressão.

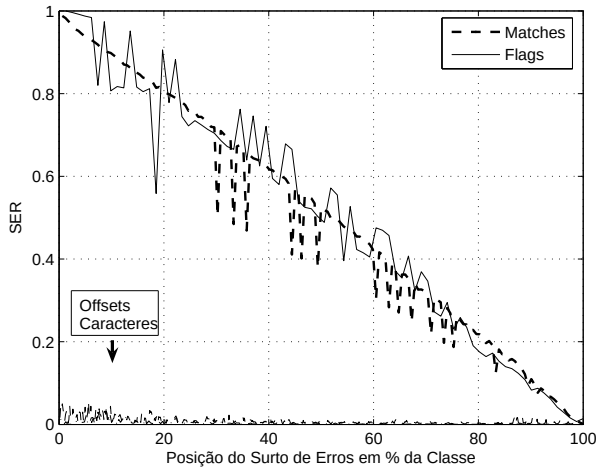

Fig. 6. Efeito do surto de erros de 48 bits no processo de descompressão, em termos da SER para cada classe em função da posição do surto de erros para o arquivo compactado a partir de 'paper4.txt'.

IV. Esquema de Proteção Desigual para LZSS

O esquema UEP proposto neste trabalho, faz uso da estrutura modificada vista na Figura 4. Os *flags*, *offsets*, *matches* e caracteres são codificados separadamente, tal que possam ter diferentes níveis de proteção em cada uma das quatro classes. Sejam t_f, t_o, t_m, e t_c os níveis do correção de erros do código de canal usados para os *flags*, *offsets*, *matches* e caracteres, respectivamente. Então, de acordo com a análise apresentada na seção anterior, um esquema de proteção desigual de erros eficiente apresenta t_f e t_m muito maiores que t_c e t_o:

$$(t_f; t_m) >> (t_o; t_c).$$

Na estratégia de proteção desigual de erros proposta neste trabalho é usado um código corretor de erros Reed-Solomon com alta taxa. A Figura 7 mostra a estrutura final do dado compactado com a inserção da redundância. Note que há um decréscimo na taxa de compressão devido à inserção das redundâncias $\widetilde{\mathbf{F}}$, $\widetilde{\mathbf{O}}$, $\widetilde{\mathbf{M}}$, e $\widetilde{\mathbf{C}}$. Este decréscimo depende da quantidade de redundância inserida em cada classe.

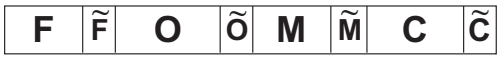

Fig. 7. Estrutura geral do arquivo compactado depois da aplicação da proteção desigual.

Uma desvantagem deste esquema UEP é que as estruturas propostas nas Figuras 4 e 7 requerem que os dados sejam colocados em *buffer* antes do armazenamento ou transmissão, o que não é requerido pelo algoritmo LZSS original [6], [9]. Contudo, a utilização do *buffer* é necessária também para o caso de proteção uniforme de erros onde códigos de bloco como os códigos Reed-Solomon são usados, pois a redundância é inserida no fim do arquivo compactado.

V. Resultados de Simulações

Os códigos Reed-Solomon usados neste trabalho são definidos sobre GF(256) e representados pela tripla (n, k, t), onde n é o número de símbolos de saída, k é o número de símbolos de entrada, e t é a capacidade de correção do código.

Como comparação, foi considerado o caso de uma proteção uniforme de erros utilizando a estrutura original da Figura 2, onde o código RS (255,251,2) é usado sobre todos os bits do arquivo compactado. Na Figura 8 pode-se ver que o esquema é muito ineficiente, pois a SER média é de $32,93\%$, mesmo havendo um acréscimo de $1,56\%$ no tamanho do arquivo compactado devido à redundância adicional. O mau desempenho é devido ao fato que as classes mais sensíveis, os *flags* e os *matches*, não são bem protegidos. Um erro nos *flags* ou nos *matches* provoca propagação de erros que corrompem o tamanho do arquivo descompactado.

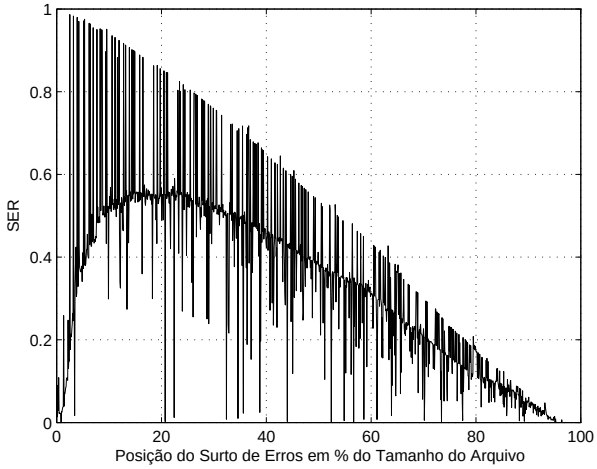

Fig. 8. Efeito do surto de erros no processo de descompressão, em termos da SER em função da posição do surto de erros, para o arquivo 'paper4.txt' usando proteção uniforme de erros com $t = 2$.

O aumento da capacidade de correção de erros do código Reed-Solomon usado no esquema de proteção uniforme para

até $t = 5$ não possibilita uma melhora significativa de desempenho, pois o surto de erros é mais longo que a capacidade de correção do código. A SER média é ainda em torno de 31%. A SER média somente decresce em um esquema de proteção uniforme quando $t = 6$. O código Reed-Solomon seria então capaz de corrigir qualquer surto simples de até $6 \times 8 = 48$ bits, porém isto gera um custo de $4,71\%$ de acréscimo no tamanho do arquivo. Um comportamento similar ocorre quando é considerado um esquema EEP com a estrutura de arquivo proposta na Figura 4. Somente com a diferença de que a SER média no arquivo descompactado é em torno de 10% para $t = 1$ até $t = 5$. A SER média reduzida é devido ao fato de que a nova estrutura proposta na Figura 4 é menos sensível à propagação de erros que a estrutura original.

Para avaliar o esquema proposto, inicialmente adicionamos proteção apenas para a classe dos bits de *flag*, utilizando o código RS (255,243,6). Na Figura 9 pode-se ver uma grande redução no efeito dos surtos de erros no processo de descompactação. A SER média é reduzida para $7,17\%$, com um acréscimo de apenas $0,31\%$ no tamanho do arquivo compactado, pois os *flags* são somente uma parte pequena do arquivo total.

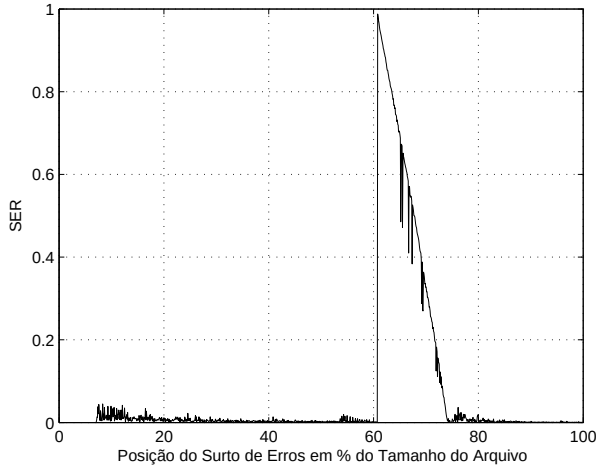

Fig. 9. Efeito do surto de erros no processo de descompressão, em termos da SER em função da posição do surto de erros, para o arquivo 'paper4.txt' usando o esquema de proteção desigual de erros para os *flags*.

A Figura 10 mostra o efeito de inserir redundância adicional para os bits de *flag* e também para os bits de *match*. Neste caso também foi usado o código RS (255,243,6) para proteger os *matches* e os *flags*. O efeito dos surtos de erros decrescem ainda mais, onde a SER média é de apenas $0,40\%$ com um acréscimo total de $0,95\%$ no tamanho do arquivo. Como os erros nos caracteres e nos *offsets* praticamente não se propagam e não geram um tamanho de arquivo errado depois da descompactação, não foi aplicado nenhuma proteção especial para estas classes.

Como validação, foram feitas simulações com vários arquivos de tipos diferentes da base de dados Calgary Corpus [22], com dois tamanhos diferentes para o surto de erros. As Tabelas II e III reúnem os resultados quando surtos de erros de 48 e

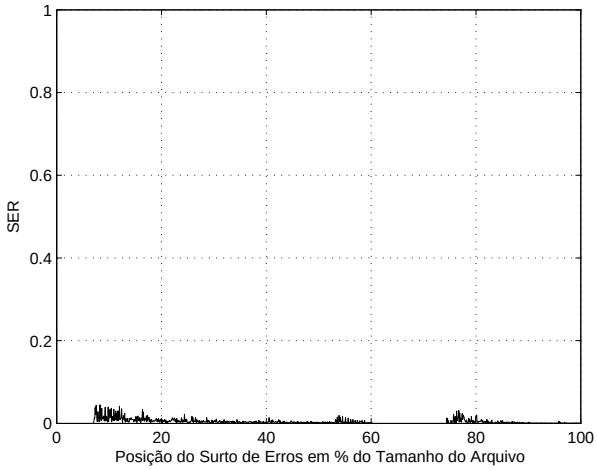

Fig. 10. Efeito do surto de erros no processo de descompressão, em termos da SER em função da posição do surto de erros, para o arquivo 'paper4.txt' usando o esquema de proteção desigual de erros para os *flags* e os *matches*.

96 bits de comprimento são aplicados para alguns arquivos de texto. Para o esquema UEP foram considerados os parâmetros ($t_f = t_m = 6$, $t_c = t_o = 0$) para o caso de um surto de erros de 48 bits de comprimento, e ($t_f = t_m = 12$, $t_c = t_o = 0$) para um surto de erros de 96 bits de comprimento. A partir das tabelas podemos observar que o método UEP proposto é capaz de reduzir consideravelmente – para menos que $0,5\%$ para o surto de 48 bits e para menos de $1,00\%$ para o surto de 96 bits – a SER média no arquivo descompactado. O custo para essa SER média muito reduzida é pequeno, sendo em torno de 1% e 2% de redundância adicional para os casos de surtos de 48 bits e 96 bits, respectivamente.

TABELA II

SER MÉDIA E REDUNDÂNCIA ADICIONAL PARA O ESQUEMA UEP PARA OS ARQUIVOS PAPER1, PAPER2, PAPER3, PAPER4 E PAPER5.

		Arquivo de Texto				
Surto	Desempenho	paper1	paper2	paper3	paper4	paper5
48 bits	SER Média	0,18%	0,16%	0,17%	0,40%	0,42%
	Redundância	1,00%	1,02%	1,00%	0,95%	0,91%
96 bits	SER Média	0,39%	0,33%	0,34%	0,84%	0,83%
	Redundância	2,16%	2,21%	2,17%	2,01%	2,00%

TABELA III

SER MÉDIA E REDUNDÂNCIA ADICIONAL PARA O ESQUEMA UEP PARA OS ARQUIVOS PROGC, PROGP E OBJ1 E PAPER6.

		Arquivo de Texto			
Surto	Desempenho	progc	progp	obj1	paper6
48 bits	SER Média	0,29%	0,37%	0,20%	0,28%
	Redundância	1,02%	1,04%	0,82%	0,98%
96 bits	SER Média	0,57%	0,73%	0,38%	0,51%
	Redundância	1,87%	2,11%	1,72%	2,12%

Por fim, o método proposto foi comparado com o método introduzido em [21] em termos de aumento no tamanho do arquivo compactado requerido para cada esquema com objetivo

de proteger os *flags* e os *matches* contra um surto de erros de 20 bits. A Tabela IV mostra os resultados para três arquivos, onde os dados correspondentes ao método em [21] são obtidos diretamente a partir de [21, Tabela I]. São considerados estes três arquivos porque eles representam classes de arquivos muito diferentes (texto em Inglês, código fonte Pascal e código Pascal compilado). A partir dessa tabela podemos observar que o esquema proposto requer muito menos redundância que o método baseado em códigos de repetição que é usado em [21]. O código Reed-Solomon usado para proteção dos *flags* e dos *matches* no caso de um surto de erros de 20 bits de comprimento é o código (255,249,3), o qual tem taxa de 0,976, enquanto o código de repetição usado em [21] é de taxa 0,5.

O método proposto em [21] apresenta uma vantagem sobre o método proposto nesse trabalho: o fato de que a sua estrutura requer um *buffer* muito menor. Porém, o método proposto neste artigo garante a proteção contra surtos de erros utilizando uma redundância muito menor.

TABELA IV

AUMENTO NO ARQUIVO COMPACTADO REQUERIDO PARA PROTEGER OS *flags* E OS *matches* CONTRA UM SURTO DE ERROS DE 20 BITS.

Arquivo de Texto	paper1	progp	obj1
Método Kitakami e Kawasaki em [21]	14.80%	21.00%	11.92%
Esquema Proposto	0.51%	0.52%	0.40%

VI. CONCLUSÕES

Neste trabalho foi proposto um novo esquema de proteção desigual de erros para dados compactados com LZSS. O esquema proposto utiliza uma nova estrutura para o arquivo compactado, aplicando uma redundância adicional. Essa redundância adicional é conseguida utilizando um código corretor de erros Reed-Solomon.

Os resultados das simulações mostram que o esquema proposto é muito mais eficiente que os esquemas de proteção uniforme de erros. Com uma redundância adicional em torno de 2%, o esquema foi capaz de reduzir a SER média nos arquivos descompactados para menos que 1% na presença de um surto de erros de 96 bits. Além do mais, comparado com outro método conhecido na literatura, o esquema proposto apresentou diversas vantagens.

REFERÊNCIAS

[1] T.C. Bell, J.G. Cleary, and I.H. Witten, *Text compression*, Prentice Hall, 1990.

[2] W. K. Ng, and C. V. Ravishankar, "Block-oriented compression techniques for large statistical databases", IEEE Trans. on Knowledge and Data Engineering, vol. 9, no. 2, pp. 314–328, 1997.

[3] S. W. Chiang, and L. M. Po, "Adaptive lossy LZW algorithm for palettised image compression", IEE Electronics Letters, vol. 33, no.10, pp. 852–854, 1997.

[4] D. Greene, M. Vishwanath, F. Yao, and T. Zhang, "A progressive Ziv-Lempel algorithm for image compression", Proceedings of Compression and Complexity of Sequences, 1997.

[5] J. Ziv, and A. Lempel, "A universal algorithm for sequential data compression", IEEE Transactions on Information Theory, vol. 23, pp. 337-343, 1977.

[6] J. A. Storer, and T. G. Syzmanski, "Data Compression via Textual Substituion", Journal of the ACM, vol. 29, pp. 928-951, 1982.

[7] J. Ziv, and A. Lempel: "Compression of individual sequences via variable-rate coding", IEEE Transactions on Information Theory, vol. 24, pp. 530-536, 1978.

[8] K. Sayood: *Introducion to Data Compression*, Morgan Kaufmann Publishers, 2000.

[9] T. Bell, "Better OPM/L text compression", IEEE Transactions on Communications, vol. 34, no. 12, pp. 1176–1182, 1986.

[10] J. C. Lo, M. Kitakami, and E. Fujiwara, "Reliable Logic Circuits with Byte Error Control Codes – A Feasibility Study", Proceedings of IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, November 1996.

[11] C. de Almeida, and R. Palazzo Jr., "Efficient two-dimensional interleaving technique by use of the set partitioning concept", IEE Electronics Letters, vol. 32, no. 6, pp. 538–540, 1996.

[12] W. Coene, H. Pozidis, M. Van Dijk, J. Kahlman, R. Van Woudenberg, and B.Stek, "Channel coding and signal processing for optical recording systems beyond DVD", IEEE Transactions on Magnetics, vol. 37, no. 2, pp. 682–688, 2001.

[13] E. Biglieri, J. Proakis, and S. Shamai, "Fading channels: information-theoretic and communications aspects", IEEE Transactions on Information Theory, vol. 44, no. 6, pp. 2619–2692, 1998.

[14] W. Zhu, and J. Garcia-Frias, "Stochastic context-free grammars and hidden Markov models for modeling of bursty channels", IEEE Transactions on Vehicular Technologie, vol. 53, no. 3, pp. 666–676, 2004.

[15] F. J. MacWilliams, and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, 1977.

[16] S. Lin and D.J. Costello Jr., *Error Control Coding*, Prentice-Hall, 2004.

[17] B. Masnick and J. Wolf, "On linear unequal error protection codes", IEEE Transactions on Information Theory, 1967, vol. 13, no. 4, pp. 600–607, 1967.

[18] E. Fujiwara and M. Kitakami, "Unequal error protection in Ziv-Lempel coding", IEICE Transactions on Information & Systems, vol. E86-D, no. 12, pp. 2595–2600, 2003.

[19] M. Kitakami, and S. Nakamura, "Burst error recovery for Huffman coding", IEICE Transactions on Informations. & Systems, vol. E88-D, no. 9, pp. 2197–2200, 2005.

[20] M. A. A. Siqueira, M. E. Pellenz, R. D. Souza, and A. O. Santin, "Esquema de Proteção Desigual de Erros para Dados Compactados usando LZSS". Anais do XXII Simpósio Brasileiro de Telecomunicações SBT'05, 2005.

[21] M. Kitakami, and T. Kawasaki: "Error Recovery Method for LZSS Coding", Proceedings of International Symposium on Theory and its Applications, Parma, pp. 1158–1163, October 2004.

[22] http://www.data-compression.info/Corpora/CalgaryCorpus/

[23] S. Perkins and D. H. Smith, "Robust data compression: variable length codes and burst errors", The Computer Journal, vol 48, no.3, pp. 315–322, 2005.

[24] D. Maniezzo, M. Cesana, P. Bergamo, M. Gerla, and K. Yao, "Real-time caption streaming over WiFi network", Proceedings IEEE International Conference on Information Technology: Research and Education, 2003.

[25] Y. Zhao, X.Zhang, R-S. Hu, J. Xue, X. Li, L. Che, R. Hu, and L. Schopp, "An automatic captioning system for telemedicine", Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing, 2006.

[26] E. Fujiwara, *Code Design for Dependable Systems: Theory and Practical Applications*, John Wiley and Sons Ltd, 2006.

[27] T. Okuda, E. Tanaka, and T. Kasai, "A method for correction of grabled words based on the Levenshtein metric", IEEE Transactions on Computer, vol. C-25, pp. 172–176, 1976.

[28] R. Bauer, and J. Hagenauer: "On variable length codes for iterative source/channel decoding", Proceedings of IEEE Data Compression Conference, 2001.